CSC411: Project #1

Due on Monday, January 29, 2018

Yuchen Wu

January 28, 2018

Part 0

Introduction to CSC411 Project 1 Report and Instructions for Reproducing the Results

The submitted files for this project include a zip file named `faces.zip`; a pdf document named `faces.pdf` (this report); a Python script named `faces.py`; and a Latex source file named `faces.tex`

The zip file contains all the cropped out images downloaded from the FaceScrub dataset and used for completing project 1. It also contains figures and images used in this report. Please put the `figures` folder under the same directory with `faces.tex` to regenerate this pdf report if needed.

For reproducing the results, please first extract the zip file and put the `cropped_images` folder under the same directory with the Python script, `faces.py`. All Python functions mentioned in this report can be found in the Python script, `faces.py`. Inside the `if __name__ == "__main__"` block, there are 10 functions (`part_1` to `part_8`). The returned values or output of each functions produces the result for the corresponding part of the project. Please feel free to comment or uncomment any function inside this block to reproduce the result of certain parts.

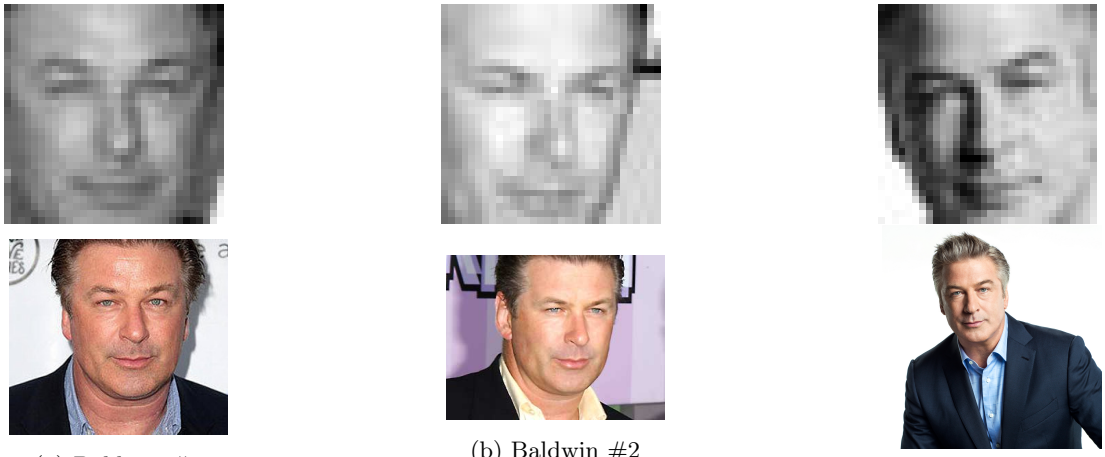
Note that function `part_1` in `faces.py` is used for downloading images from FaceScrub dataset, cropping the images and resizing it to 32 by 32 pixels. To run this function, please put the two text documents named `facescrub_actors.txt` and `facescrub_actresses.txt` under the same directory with the Python script, `faces.py`.

Part 1

Dataset Description

Figure 1 to Figure 3 provides nine examples of the images in the dataset and the corresponding cropped out faces.

Generally speaking, the dataset has decent annotation quality. The bounding boxes for most images are accurate and the cropped out faces are aligned with each other (such as for Baldwin #1, #2 and #3, as shown below in figure 1). However, for a few images, the bounding boxes seems to be too small or too large (such as cropped out faces shown in figure 3). In addition, for images in which the actor or actress is not facing the camera, the cropped-out faces are also sometimes not completely aligned (such as Baldwin #31 and #44 shown in figure 2). There are also a small number of cropped images (around 1% in the dataset) that do not show faces, which is either because the original image does not include any face (such as Carell #39 and Chenoweth #13) or the bounding box is incorrect (such as Bracco #89), as shown in figure 4.



(a) Baldwin #1 (b) Baldwin #2 (c) Baldwin #3

Figure 1: Examples of Correctly Cropped out Faces

(a) Baldwin #31 (b) Baldwin #44

Figure 2: Examples of Cropped out Faces that are not Aligned with Most Faces

(a) Baldwin #34

(b) Drescher #126

(c) Radcliffe #129

Figure 3: Examples of Cropped out Faces that are too Small or too Large

(a) Bracco #89

(b) Carell #39

(c) Chenoweth #13

Figure 4: Examples of Error Cropped out Faces

Part 2

Dataset Separation Algorithm

According to the requirement, 90 images are needed in total for each actor (the training set requires 70 face images per actor/actress, the validation set requires 10 face images per actor/actress, and the test set requires 10 face images per actor/actress). Due to difference in total number of images for each actor/actress downloaded from the FaceScrub dataset, the algorithm first selects 90 images randomly among downloaded images of each actor/actress. Then, it randomly chooses 10 images and another 10 images from the 90 images of each actor and put them into the *validation* set and the *test* set, respectively. The rest of images are put into the *training* set. Randomization is used in this algorithm to avoid potential bias. This algorithm was implemented in function `part_2`. The function returns 3 dictionaries storing the separated training set, validation set and test set, respectively. (Please run function `part_2` in `faces.py` to see its returning examples.) The values in each dictionary are the filenames of the selected pictures and the corresponding keys are labels (i.e. name of corresponding actors/actresses).

The separated sets represented in dictionaries obtained from function `part_2` can then be converted to corresponding feature matrices ($\mathbf{X}$) and label matrices ($\mathbf{Y}$) through helper function `create_set` when needed. The function `create_set` takes two input parameters. The first parameter is one of the sets (training, validation or test) returned from `part_2` and the second parameter is another dictionary that stores the value of labels to be used (e.g. -1&1 or 0&1 for a binary classifier). The function `create_set` is explicitly called in function `part_3_classifier` to get the feature matrices ($\mathbf{X}$) and label matrices ($\mathbf{Y}$) of training, validation and test set. Please run `part_3_classifier` and see what it returns if needed.

Part 3

Binary Classification Using Linear Regression: Distinguish Pictures of Alec Baldwin from Pictures of Steve Carell

The Cost function

Linear regression is used, as required, to build the classifier. The hypothesis $h_\theta(x)$ is

$$\begin{aligned} h_\theta(x) &= h_\theta(x_0, x_1, x_2, ..., x_{1024}) \\ &= x_0\theta_0 + x_1\theta_1 + x_2\theta_2 + ...x_{1024}\theta_{1024} \\ &= \boldsymbol{\theta^T}\mathbf{x} \end{aligned}$$

where $x_0 = 1$ and x_k is the k^{th} pixel of the picture. Since each image after being cropped and resized is 32 by 32, so there are 1024 pixels in total for each image (i.e. $k = 1$ to 1024). In addition, the input images are divided by 255.0 so that all inputs are in the 0 to 1 range.

The cost function chosen to minimize was the sum of squared differences between the expected outputs and the actual outputs,

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

(Note: The expected output (label) for Alec Baldwin is -1 and 1 for Steve Carell. m is the number of inputs.)

Performance of the classifier

Value of cost function on the training set: $f(x)_{training\ set} = 0.000374 \pm 0.000044$

Value of cost function on the validation set: $f(x)_{validation\ set} = 0.207773 \pm 0.042181$

Performance of the classifier on the training set: $100.0\% \pm 0.000000\%$

Performance of the classifier on the validation set: $90.00\% \pm 5.477226\%$

(Note: this result was produced by function `part_3` and was obtained from 5 trials with different training and validation sets in same size.)

Code of the function used for computing the output of the classifier

The code of the function used to compute the output of the classifier is shown below (`faces.py` from line 204 to 211):

```
def output(x,theta,anti_classify_act):
"""
Compute the output of the Binary Classifier with a threshold of 0
Take:
    x: input
    theta: trained weights
    anti_classify_act: a dictionary that stores the correct label of each actor
        for this part: {-1 : 'Alec Baldwin', 1:'Steve Carell'}
Return:
    result: output of the classifier i.e. name of the corresponding actor
"""
    result = []
    for k in np.matmul(theta,x):
        if k > 0 : # if h(x)>0 then classify as 'Steve Carell' for this part
            result.append(anti_classify_act[1])
        else: # otherwise classify as 'Alec Baldwin' for this part
            result.append(anti_classify_act[-1])
    return result
```

To classify the faces, a threshold of 0 is used to differentiate the two classes.

$$Output(x) = \begin{cases} -1, h_\theta(x) < 0 \\ 1, h_\theta(x) > 0 \end{cases}$$

In this case, -1 corresponds to "Alec Baldwin" and 1 corresponds to "Steve Carell".

Parameter selections
The initial value of θs, learning rate α and stopping condition EPS for gradient descent were selected carefully in order for the system to work. Since in this part, the system is only expected to work and over-fitting issues are not yet considered, maximum number of iteration for gradient descent process was set to a large number (100000) to avoid early stopping.

- Select initial value of θs:
  Since the cost function is convex. It is expected that initial value of θs are selected such that the initial value of the cost function is relatively small. In other words, it is expected that the initial value of θs are close to the desired value so that the cost function could converge to global minimum quickly during gradient descent process.
  Table 1 below shows the initial value of function with initial θs under normal distribution with different variance. The result was generated from function `part_3`. Note: there were 5 trials for each initial distribution of θs.

Table 1: Initial Value of Cost Function under Different Initial Value of θs

Initial θs	Initial value of the cost function
$N(0, 0.0)$	140.00 ± 0.00
$N(0, 0.0001)$	139.98 ± 0.02
$N(0, 0.001)$	139.95 ± 0.32
$N(0, 0.01)$	142.38 ± 4.35
$N(0, 0.1)$	464.42 ± 273.86

  Based on the result, all initial θs were selected to be zeros because it, in general, minimized the initial value of the cost function.

- Select learning rate α for gradient descent:

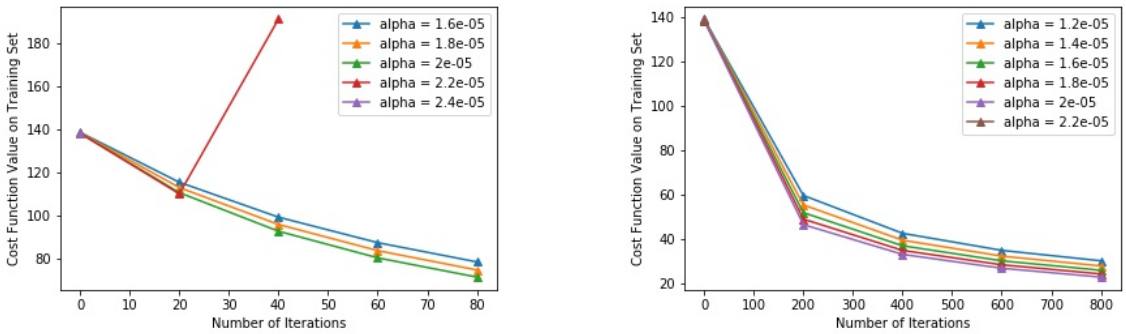


Figure 5: Value of Cost Function vs. Number of Iterations

The learning rate α is expected to be small enough such that the cost function could converge to its minimum after performing gradient descent. However, it also should not be too small because in that

case it would lead to slow convergence. Figure 5 shows the change in value of the cost function during gradient descent with different learning rates. (Note: The initial θs were all set to be zero based on previous observations.)

The result shows that the system diverges when learning rate is greater than 2×10^{-5}. The rate of convergence does not vary a lot for the learning rate α ranging from 1.2×10^{-5} to 2×10^{-5}. Therefore, the learning rate was eventually set to 1.5×10^{-5} to ensure convergence and maximize rate of convergence to a certain extent.

- Select stopping condition for gradient descent (EPS):
  The stopping condition EPS was selected to be a fixed number, 1e-5, which was given in function `grad_descent` provided in lecture.

Part 4

(a) View $(\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_n)$ ***as an Image***

θs (except θ_0) were reshaped to (32,32) for visualization. Figure 6 below shows the θs obtained by training using the full training dataset (70 images for each actor).

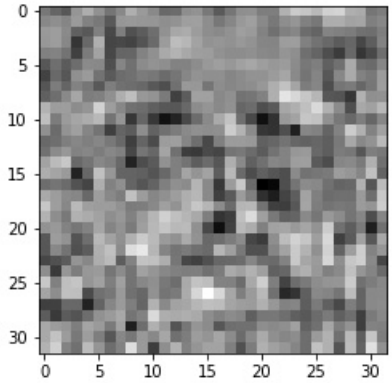


Figure 6: θs from Full Training Set

Figure 7 below shows the θs obtained by training using a training set that contains only two images of each actor.

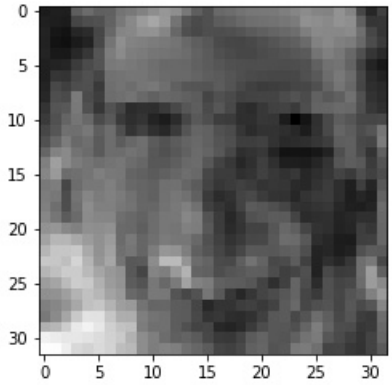


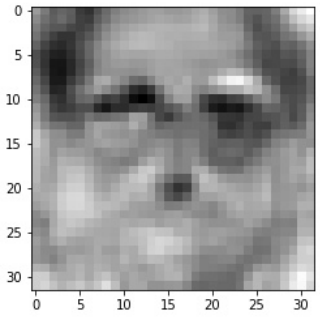
Figure 7: θs from Training Set Containing Two Images of each Actor

Note: For this part, all parameters used are same as those in Part 3. The maximum iterations for gradient descent were set to be 100000 (a fairly large number) to avoid early stopping.

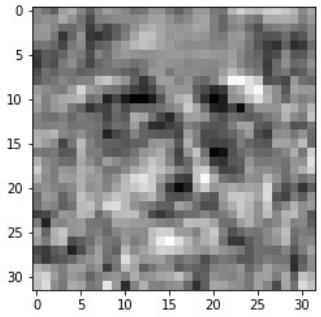
(b) Produce both kinds of Visualizations

Using the full training set with same setup (stopping condition EPS = 1e-5, learning rate α=1.5e-5), the two kinds of visualizations can be produced by stopping the Gradient Descent process earlier and later in the process or initialing θs using different strategies.

- Stopping the Gradient Descent process earlier and later in the process.
  Figure 8 (left) shows the image of θs when the gradient descent process was stopped at 100^{th} iteration. Figure 8 (right) shows the image of θs when the gradient descent process was stopped at 10000^{th} iteration. All θs were initialized to be zero.



(a) Earlier (100th iteration)



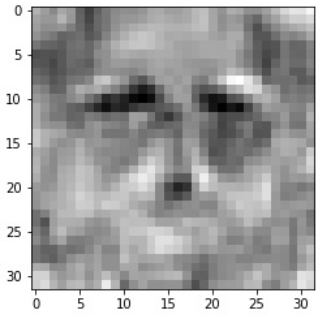
(b) Later (10000th iteration)

Figure 8: Stopping the Gradient Descent process earlier and later

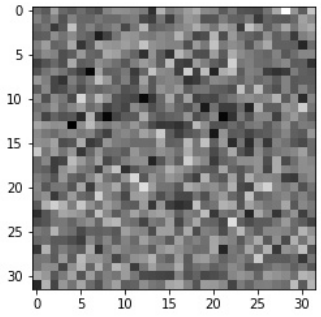
The result shows that, when all the θs are zero initially, the image of θs after being trained tends to look more like a face when the gradient descent process stops relatively earlier.

- Initializing the θs under normal distribution with different standard deviation.

  Figure 9 (left) shows the image of θs when the initial values of θs are under normal distribution with 0.0001 standard deviation and 0 mean; Figure 9 (right) shows the image of θs when the initial values of θs are under normal distribution with 0.1 standard deviation and 0 mean. The maximum number of iteration was fixed at 1000.



(a) $\mu = 0$ $\sigma = 0.0001$



(b) $\mu = 0$ $\sigma = 0.1$

Figure 9: Initialize values of θs under normal distribution with different mean and variance

It can be seen from the two images that, when the gradient descent stops early in the process (i.e.

early stopping), the image tends to look like a face when the standard deviation of initial θs are small, which means that θs are very close to a constant (zero in this case) initially.

Part 5

Binary Classification Using Linear Regression: Classify the Actors as Males or Females
For this part, the training set contains 70 images of each actor/actress and the validation set contains 10 images of each actor/actress listed below:

- Females: 'Lorraine Bracco', 'Peri Gilpin', 'Angie Harmon'
- Males: 'Alec Baldwin', 'Bill Hader', 'Steve Carell'

The test set contains 10 images of each actor/actress listed below:

- Females: 'Kristin Chenoweth', 'Fran Dresher', 'America Ferrera'
- Males: 'Gerard Butler', 'Daniel Radcliffe', 'Michael Vartan'

Females are labelled as -1 and males are labelled as 1.

Over-fitting Demonstration
Figure 10 shows the performance of the classifiers on the training and validation sets as well as on the test set verses the size of the training set. (10 trials were run for each different training set size with different images in the training set.)
Note that for this part, the parameters (stopping condition EPS, learning rate α and initial values of θs) were selected in the same way as in Part 3. The maximum number of iteration was set to a fairly large number (10000) to avoid early stopping. It was also noticed that stopping earlier in the gradient descent could sometimes prevent the model from over-fitting to the training set. This observation is discussed in detail in Part 7.

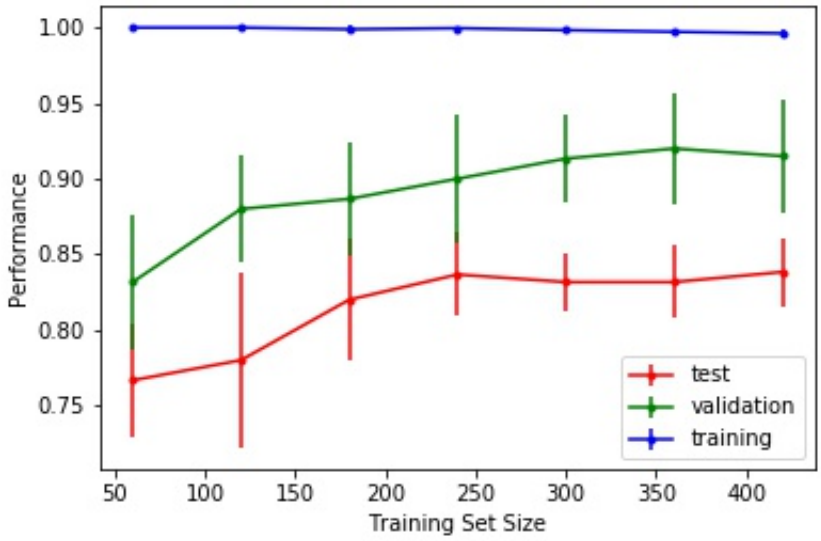


Figure 10: Performance of the classifiers on the training, validation and test set vs. different training set size ($\alpha = 0.000005$; $EPS = 1e^{-}5$; Initial $\theta = 0s$; $Max\ Num.\ of\ Iteration = 10000$)

The above result shows that the classifier always has highest accuracy on the training set (nearly 100%), although the performance decreases slightly as the size of training set grows. Although performance of the classifier on the validation set is always higher than which on the test set, performance on both sets

gets improved as the size of training set becomes larger. The result indicates that, when the training set is relatively small, the model tends to be over-fitting to the training set because its performance on the validation set (80%-85%) and test set (75%-80%) is much worse than on the training set. As the training set size becomes larger (about 70 images for each actor), performance on the validation set increases to around 90%, which indicates that the model is less over-fitting to the images used in the training set. However, since the performance on the test set (i.e. another six actors) is much less than on the training set (80%-85%), the model is still over-fitting to the gender of the six actors/actresses used in the training set.

Report the performance on the test set

The performance of the classifier on the 6 actors who are not included in *act* (i,e performance on the test set as shown above) is $83.8333\% \pm 2.2423\%$ when using the full training set (70 images for each actor, 420 in total). The performance would decrease when using training set with smaller size.

Part 6

(a) Compute $\dfrac{\partial J}{\partial \theta_{pq}}$

As required, The cost function is still the sum of squared differences between the expected outputs and the actual outputs:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{m}(\sum_{j=1}^{k}(\theta^T x^{(i)} - y^{(i)})_j^2)$$

So, the partial differentiation with respect to θ_{pq} can be computed by chain rule,

$$\begin{aligned}
\frac{\partial J}{\partial \theta_{pq}} &= \frac{\partial(\sum_{i=1}^{m}(\sum_{j=1}^{k}(\theta^T x^{(i)} - y^{(i)})_j^2))}{\partial \theta_{pq}} \\
&= \sum_{i=1}^{m} \frac{\partial(\sum_{j=1}^{k}(\theta^T x^{(i)} - y^{(i)})_j^2)}{\partial \theta_{pq}} \\
&= \sum_{i=1}^{m} 2 \cdot (\sum_{h=1}^{n} \theta_{qh}^T x_h^{(i)} - y_q^{(i)}) \cdot \frac{\partial(\sum_{h=1}^{n} \theta_{qh}^T x_h^{(i)} - y_q^{(i)})}{\partial \theta_{pq}} \\
&= 2 \cdot \sum_{i=1}^{m}(\sum_{h=1}^{n} \theta_{qh}^T x_h^{(i)} - y_q^{(i)}) \cdot x_p^{(i)} \\
&= 2 \cdot \sum_{i=1}^{m}(x_p^{(i)} \cdot (\sum_{h=1}^{n} \theta_{qh}^T x_h^{(i)} - y_q^{(i)}))
\end{aligned}$$

Note: the dimension of θ, $x^{(i)}$ and $y^{(i)}$ were mentioned in the project instructions.

(b) Derivative of J(θ) with respect to all the components of θ

Define m as the number of training examples; $n = 1025$ as the number of features (pixels) of each image with an additional "1"; k as the number of possible output of the model. Then, $\boldsymbol{\theta}$ is a $n \times k$ matrix, $\mathbf{X}$ is a $n \times m$ matrix and $\mathbf{Y}$ is a $k \times m$ matrix.

Method 1

It can be shown that

$$\begin{aligned}
(2X(\theta^T X - Y)^T)_{pq} &= 2 \cdot \sum_{i=1}^{m} X_{pi}(\theta^T X - Y)^T_{iq} \\
&= 2 \cdot \sum_{i=1}^{m} X_{pi}(\theta^T X - Y)_{qi} \\
&= 2 \cdot \sum_{i=1}^{m} X_{pi}((\theta^T X)_{qi} - Y_{qi}) \\
&= 2 \cdot \sum_{i=1}^{m} X_{pi}(\sum_{h=1}^{n} \theta^T_{qh} X_{hi} - Y_{qi}) \\
&= 2 \cdot \sum_{i=1}^{m} (x^{(i)}_p \cdot (\sum_{h=1}^{n} \theta^T_{qh} x^{(i)}_h - y^{(i)}_q)) \\
&= \frac{\partial J}{\partial \theta_{pq}} \text{ (Computed in Part 6 (a))}
\end{aligned}$$

Therefore, the derivative of $J(\theta)$ with respect to all the components of θ can be written in matrix form as $2X(\theta^T X - Y)^T$.

Method 2

The cost function $J(\boldsymbol{\theta}) = \sum_{i=1}^{m}(\sum_{j=1}^{k}(\theta^T x^{(i)} - y^{(i)})^2_j)$ can be written in matrix form as

$$tr((\theta^T X - Y)(\theta^T X - Y)^T) = tr(\theta^T X X^T \theta - Y X^T \theta - \theta^T X Y^T + Y Y^T)$$

According to general formula of matrix differentiation, the gradient of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ is:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = 2XX^T\theta - 2XY^T = 2X(X^T\theta - Y^T) = 2X(\theta^T X - Y)^T$$

(c) Implement the cost function from Part 6(a) and its vectorized gradient function

The code below shows the implementation of the cost function from Part 6(a) and its vectorized gradient function in Python. (`faces.py` from line 645 to 651)

```
def part_6_f(x, y, theta):
     """
     Quadratic loss function for part 6.
     Take:
          x: the input data (images)
          y: the label of input data
          theta: the weight matrix
     Return:
          Quadratic loss function derived in (b) method 2
     """
    cost = np.matmul(theta.transpose(),x)-y
    #the cost function in matrix form as derived in (b) Method 2
    return np.trace(np.matmul(cost,cost.transpose()))

def part_6_df(x, y, theta):
     """
     Compute the gradient of the quadratic loss function for part 6 wrt thetas.
     Take:
          x: the input data (images)
          y: the label of input data
          theta: the weight matrix
     Return:
          The gradient matrix of loss function wrt thetas derived in (b)
     """
    cost = np.matmul(theta.transpose(),x)-y
    return 2*np.matmul(x,cost.transpose())
```

(d) Compute the gradient using finite-difference approximations

Code for computing the gradient components using finite differences

Code shown below is used for computing the gradient components using finite differences and compare them to the gradient computed using function in Part 6(b) (`faces.py` from line 690 to 712). Relative error is used to compared the approximated values to the output of the gradient function. The formula for relative error is

$$Err(Approx, Output) = |Approx - Output|/(|Approx| + |Output|).$$

```
def part_6d_finite_diff_approx(f,x,y,theta,h,p,q):
     """
     Compute the gradient component theta_pq
     Take:
      f: the cost function    x: input features    y: labels    theta: weights
     Return:
          the partial derivative computed using finite difference approximation
               wrt theta_pq
     """
    thetah = theta.copy()
    # increase theta_pq by h
    thetah[p][q] += h
    return (f(x, y, thetah)-f(x, y, theta))/h
def part_6_d_error(f,x,y,theta,h,p,q):
     """
     Compare the error of the pq component of gradient wrt theta between the finite
          difference approximation and value obtained from function in Part 6b
     Take:
          f: the cost function    x: input features    y: labels
          theta: weights    h: finite difference
          p and q: coordinates
     Return:
          error: Relative error between approximation and the actual value
     """
    approx = part_6d_finite_diff_approx(f, x, y, theta,h,p,q)
    actual = part_6_df(x, y, theta)[p][q]
    #Compute the relative error between approximation and the actual value
    #The formula for relative error is error(a,b)=|a-b|/(|a|+|b|)
    error = np.abs(approx-actual)/(np.abs(approx)+np.abs(actual))
    return error
```

When h is set to 10^{-8}, the mean relative error between the approximated gradient and the gradient obtained from formula in (b) is around 10^{-7}, which indicates that the vectorized gradient function works. The reason for selecting h to be 10^{-8} is discussed below.

Select the Value of h

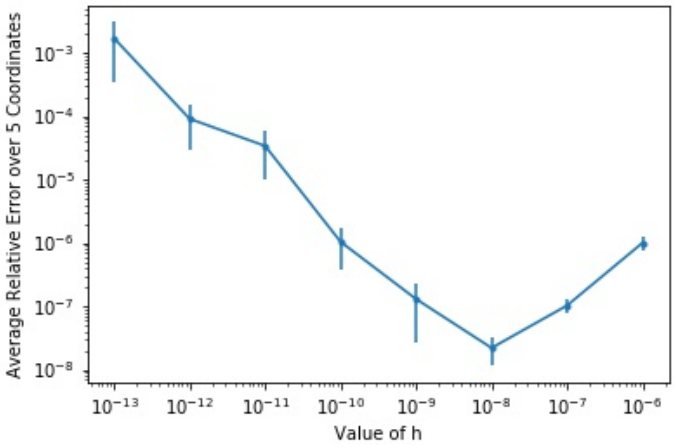


Figure 11: Relative Error between Finite Difference Approximation and True Gradient

Figure 11 above shows the relative error, computed using function `part_6_d_error` above, between the gradient computed using the function in Part 6(b) and the finite-difference approximations along 5 coordinates with different values of h. From the plot, the relative error between finite-difference approximation and the true gradient is minimized when h is around 10^{-8}. Although based on definition, the error should converge to zero as h goes to zero, it ,in fact, increases when h becomes too small due to numerical issues (e.g. arithmetic underflow).

Part 7

Multi-class Classification Using Linear Regression: Run Gradient Descent on the Set of Six Actors/Actresses

Parameter Selections

Similar to Part (3), the initial values of θs, learning rate α and stopping condition EPS for gradient descent were first selected carefully in order for the system to work. For this part, the stopping condition EPS was still chosen to be 1e-5, which was given in function `Grad_Descent` provided in lecture. Based on result in Part 3, the initial value of θs were chosen to be zeros because that made the initial value of the cost function to be relatively small (i.e. close to the minimum). Also, for the learning rate, several values were tested and shows in figure 12 below.

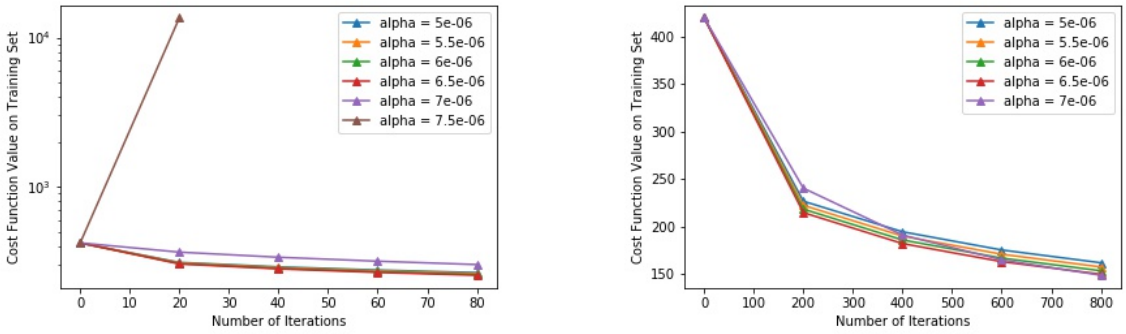


Figure 12: Value of Cost Function vs. Number of Iterations

The result shows that the system diverges when learning rate is greater than 7×10^{-6}. The rate of convergence does not vary a lot for the learning rate α ranging from 5×10^{-6} to 6.5×10^{-6}. Therefore, the learning rate was eventually set to 6×10^{-6} to ensure convergence and maximize rate of convergence to a certain extent. Another important parameter is the number of iterations. Consider that the model may be over-fitting to

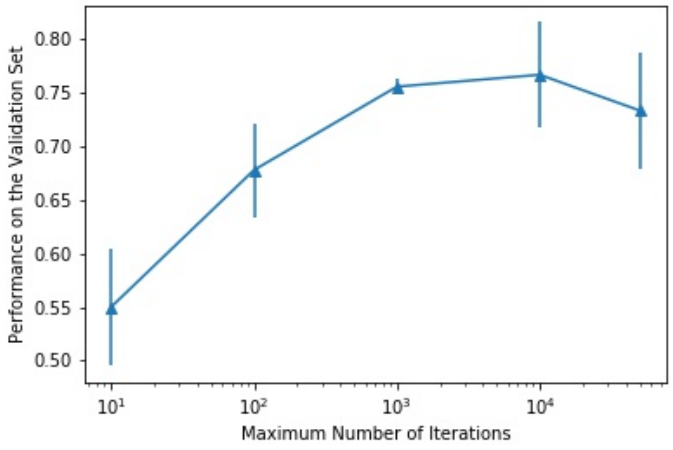


Figure 13: Performance of the classifiers on the validation set vs. number of iterations

the training set, the Gradient Descent process is expected to stop when the classifier gets best performance on the validation set rather than when the cost function is minimized. Figure 13 shows the performance of

the classifiers the validation set after different number of iterations. It shows that the performance of the classifiers on the validation set is optimized at around 10000 iterations.

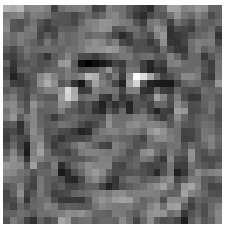
Performance of the classifier
Performance of the classifier on the training set: $98.8889\% \pm 0.2970\%$
Performance of the classifier on the validation set: $76.6667\% \pm 4.9065\%$
Note: this result was produced from function `part_7` and was got through 3 trials with different training set (size = 420) and validation set (size = 60).

Output of the Model
To produce the output of the model, firstly, the maximum value among the output vector of the hypothesis function $h_\theta(x)$ is considered as 1, other values are considered as 0. Then, the vector containing only 1 and 0s is compared with the label of each actor and the actor with the same label vector is the output of the model. For example, assume that the output of the hypothesis function is [0.6,0.8,0.3,0.5,0.2,0.6]. The maximum value is 0.8 in this vector so that the output of the hypothesis function is considered as [0,1,0,0,0,0]. Then, the actor with label [0,1,0,0,0,0] is the output of the model.

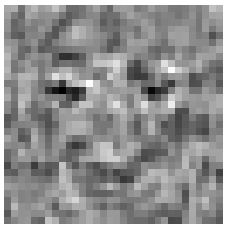
Part 8

Visualization of θs for Multi-class Classification

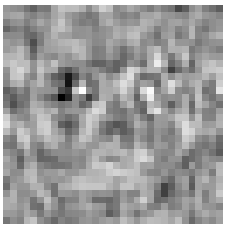
Figure 14 shows the images of each row of θs and the corresponding actors/actresses.



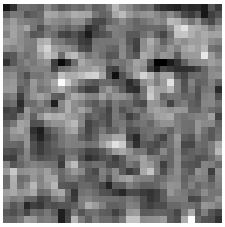
(a) Angie Harmon



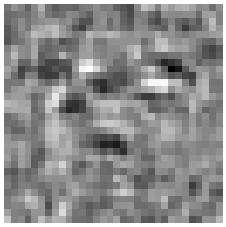
(b) Lorraine Bracco



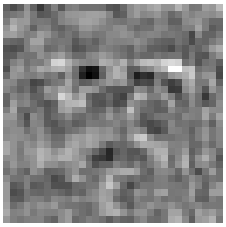
(c) Peri Gilpin



(d) Alec Baldwin



(e) Bill Hader



(f) Steve Carell

Figure 14: Parameter Visualization for each Actor/Actress